Realtime publishers

The Shortcut Guide™ To

Large Scale Data Warehousing and Advanced Analytics

sponsored by

aster data
more data. big insights.

Mark Scott

Copyright Statement

Chapter 3: Operational Considerations for Large Scale Enterprise Data Warehouses

Large scale data warehouses typically import and manage large amounts of data. Systems that need to import and store millions of rows each day and then need to report back on them have very unique challenges. This chapter will address the system considerations in meeting those challenges. There are four primary challenges on which we will concentrate:

- Importing and transforming large amounts of data on a daily basis within tight maintenance windows
- Dealing with the bandwidth and bottleneck issues on storing and retrieving large amounts of data from even larger data stores
- Optimizing queries against large data sets to provide optimized results
- Monitoring systems to assure continued operation

Data Import

Data warehouses grow large as organizations add new data sources and as the data sources themselves grow in the amount of data they host. They may come from a variety of sources: ERP, CRM, manufacturing, HR, and other related systems. Often, they have specific high volume systems that become the major daily contributors, such as credit card transactions or cell phone call logs. Identifying the sources and their issues is the first step to dealing with the challenge.

The second issue deals with the extraction transformation and load (ETL) process. This process can be very resource intensive. You must plan to control utilization of resources on both the source systems and within the data warehouse itself. In order to plan this properly, you need to consider the impact on the entire system and ensure that the source systems and the data warehouse can handle the load imposed by the load process without interfering with the users who rely on the source systems and the data warehouse.

Data Import Challenges

The first task for the operations team is to analyze the source systems and understand when they can export data. The extraction process cannot be allowed to interfere with the normal operation or slow down the users. In a global economy with many systems operating 24 hours a day, this goal can be difficult to accomplish. The other issue is that many systems have specific windows during which they can export data. Systems, such as accounting software, often have an end-of-day reconciliation. These systems have a specific point during which they can export their daily loads. This will require the extraction process to be carefully scheduled so that the data is exported at the right time without negatively impacting the ability of the source system to service users.

The challenges multiply with large-scale data warehouses. The systems often import data from a number of source systems. Each source system will have a defined time period during which it can export data. With more systems to coordinate, the effort becomes more challenging. The ETL system must be able to handle the load of multiple source systems quickly while imposing the least possible load on those systems.

The operations team must also carefully monitor the exports. Because the windows are often tight, it can be difficult to reschedule and reload data when errors occur. Careful monitoring of the process will help the operations team ensure that the system is bringing in data in a timely and accurate manner.

> **Note**
>
> I have had to work on systems where large data loads had to be processed virtually simultaneously. The ERP, production, sales entry, and quality control systems all required exporting their data during the same processing time windows. The ETL process had to be planned so that there was sufficient capacity to stage the exports from multiple systems simultaneously and allow the source systems to export their data quickly without exceeding their allotted processing windows. We needed to design a system that allowed the source systems to export as quickly as they could and then move on to other operations. We designed an ETL system that could stage data from the source systems simultaneously in the required extraction window.

The daily import system must provide flexible scheduling of the data sources, the ability to monitor the loads, and enough capacity to compensate when loads need to be rerun. When considering what systems to use when choosing ETL systems, these factors should be prime considerations. The parallel nature of these requirements naturally fits with massively parallel ETL solutions.

ETL Processes

As the name indicates, there are three processes that make up ETL: extracting, transforming, and loading. One option is for the data to land on a hard drive on the source system and then be moved to the transformation staging area. For some systems, the extracted data is written to the transformation staging area. To maintain throughput on the source system in this scenario, the transformation system must be able to effectively receive the data produced by the source system. Often, the ETL process can extract the data directly from the data source. This will place some I/O load on the source database itself, but it will keep the transformation load away from the source system, allowing it to perform other work. In other data warehouse systems, the source system itself must prepare the data for export in order to ensure that the data is properly extracted. For these source systems, it is very important that the receiving system can handle the data output as effectively as possible. This is a situation in which the order of operations changes to extract, load, and transform (ELT). An ELT approach loads data into the relational database and then applies transformations, possibly using SQL and the RDBMS' procedural language.

The implications for maintaining both ETL and ELT systems are that the target landing disks must have sufficient space to receive the target exports. Constraints in network connectivity may also interfere with the export. ELT transformations must be applied in a manner that does not place undo load on source systems, for example by executing complex logic to construct a single export file instead of multiple files that would be merged in a later step of the process. All these factors must be considered in the equation when developing and operating the extraction portion of the process.

The transformation process can consume considerable computer resources. Determining what resources are used is crucial to maintaining a balanced process. Often, the source system can play a strong role in the transformation process. One must carefully consider how much additional load that places on the source system and whether there is sufficient capacity for the source system to play this role. For example, a source system with marginal additional CPU capacity may not be able to complete even a subset of the required transformations in the time allotted. In such cases, extracted data could be moved to an MPP transformation platform where the transformation operations could be sufficiently parallelized to complete the operation with the limited time window. Having sufficient resources for the transformation process to modify and reorganize the data will help keep the process balanced properly across the enterprise resources available.

> **Note**
>
> Processing ETL packages can be a hindrance for large data loads. I once worked on a system for a cell phone company. They had daily data imports of cell phone transactions that took 27 hours to process. Obviously, if you get a load every 24 hours, you are fighting a losing battle. We had to redesign the system to streamline the process so that the data could be brought into the system quickly enough to keep up with the load. Using an ETL server that prepared the data to be inserted into the database as rapidly as possible allowed us to reduce the load from 27 hours to 2 ½ hours per day.

The final consideration is the loading process. Loading data into a database consumes a considerable amount of the resources available to the server. Systems that dedicate too many resources to data loading will suffer while servicing client requests for information. If the data can be loaded more efficiently, the data warehouse can spend more resources on responding to clients and less on getting the data into its internal tables.

As alluded to earlier, one of the most efficient ways to minimize the impact of the ETL system on both the data warehouse and the source systems is to provide dedicated ETL process servers. A dedicated ETL server can help minimize load on both the source servers and the data warehouse servers.

In a system such as a large scale data warehouse with a shared nothing architecture, nodes can be dedicated to serving the ETL process. These nodes can be designed to remove the burden of extraction from the source systems, to transform data without putting pressure on any other systems, and to place the data into the data warehouse in the most efficient manner. Another advantage of shared nothing architecture is that the data can be partitioned before it is loaded. This will distribute the load of placing data into the data warehouse across all the nodes that service the data. Each node in the warehouse will spend less time actually loading data. This will leave the warehouse more responsive to the users and help minimize the impact of large data loads. The ETL nodes are integrated into the warehouse, so the entire process is often easier to monitor and maintain.

The data may need extensive processing once it is loaded into the system. If the system uses materialized views, On-Line Analytical Processing (OLAP) cubes, or data mining model maintenance, these cycles will be added as the new data is processed. The ability to move data between structures within the database is another important consideration. For the advanced analytical structures, such as the OLAP cubes and data mining models, the amount of processor power is also much higher than is often found in other ETL operations. Understanding the full complexity of the ETL workload will help the operations team plan and maintain the system and still meet the operational service level agreements (SLAs).

Monitoring and Control

As with most operational systems, the success of the system is largely dependent on the ability to monitor the operation of the system and to control the steps of every process. ETL processes can be difficult to monitor. They interact with both the source systems and the data warehouse. Integrating the monitoring of the source systems, the ETL system, and the data warehouse can be challenging and can make the difference between successful systems and nightmares.

Once a system is operational, the staff must be able to clearly see that the steps defined in the ETL process are carried out. Having clear diagnostics that indicate where each step has succeeded will help ensure the system is operating properly. When a step fails, the system must clearly identify what part of the process has failed. It should help identify the system that is at fault to minimize the amount of time it takes to correct the fault. With large scale data warehouses and the larger amount of data that they import each day, this identification becomes more critical. There is less time and less available resources to correct errors, so quick identification of the issue and correction will help the operation team keep the system delivering to the business.

Often there needs to be flexible scheduling to accommodate schedule irregularities. Business calendars can change when certain ETL processes occur, such as end of month processing. System outages can cause the normal scheduled ETL processes to fail. Being able to work with exceptions will help the operations staff keep the data warehouse up to date with minimal additional labor.

> **Note**
>
> I worked on a large mortgage processing system. The ETL process would get the daily payments into the data warehouse for reporting. The primary source system that provided the daily payments was unstable and the process frequently failed. It was important to coordinate the activities of the data warehouse system so that failed data did not appear on the reports. Careful monitoring of the condition of the data warehouse, in conjunction with the status of the ETL process, allowed us to provide consistent, accurate data to the users. The monitoring system was crucial to our success.

The more systems that are involved in the ETL process, the more difficult it is to control. There are many well-designed ETL systems available, and many of them provide extended support for monitoring the source systems as well as the data warehouse. However, minimizing the number of systems involved will always simplify the issues of monitoring and control. Large scale data warehouses that include integrated ETL systems are often easier to control and monitor. Getting data into a large scale data warehouse is one of the most important factors in its success; thus, a successful ETL subsystem that helps load the data warehouse is paramount to its success.

Data Storage and Retrieval Challenges

All systems are limited to the performance of their weakest components. Most system tuning is the process of finding these slower components and compensating for their deficiencies.

The architecture of the system is also imminently involved with meeting the challenges of system performance. A look at the differences in architecture and how they affect operations will help both get the most out of the system implemented and provide a roadmap for determining which type of system a given enterprise should consider implementing.

Data compression is also a very important issue. Balancing the cost of compressing data on hard drives versus the cost of decompressing the data during query is one of the keys to developing a high-performance data warehouse system.

Dealing with the Bandwidth and Bottlenecks

For most database systems, the bottlenecks can be expected to be found around the disk subsystems. Getting data to and from the disk drives through the host bus adapters (HBAs) and the Storage Area Network (SAN), to and from the network attached storage (NAS) or non-network storage devices, such as appliances, are typically the areas of greatest concern.

Using multiple servers helps alleviate some of the single server bottleneck whenthe data is scaled out across multiple disk arrays. Each of the several servers has its own connections to those arrays, so the HBA bottlenecks can also be scaled across multiple devices. This helps deal with some of the primary performance issues that prevent single servers from serving as hosts for these large databases.

The multi-server data warehouse systems will eventually run into their own bottlenecks, and these issues will need to be monitored and addressed as the systems continue to grow. The performance of the system will depend on the ability of the component servers to communicate with one another and operate together as a cohesive whole.

> **Note**
>
> Incorporating multiple servers does add complexity to monitoring for bottlenecks, but the end result is typically a system that can handle a much greater load than any single server.

Bandwidth plays an issue because many of the bottlenecks will occur in the interfaces between servers. Advanced connections, such as Infiniband systems (http://www.infinibandta.org/), can help deal with this. But optimizing communications and coordinating interaction between the servers will always provide improved performance. The architecture of the data warehouse system will affect how an organization should deal with these issues.

Shared All vs. Shared Nothing Architecture

Shared all and shared nothing architectures, discussed in the previous chapter, offer different challenges in terms of data storage and retrieval. Understanding where their bottlenecks are likely to occur will help optimize their operation.

Shared All Architecture

A shared all architecture uses multiple servers as peers to cooperatively work on the same data structures. As any single server can handle any task presented to the system, they can freely share the workload presented to the warehouse. The operational and performance challenges come in maintaining communication between the servers:

- Any server in the group can be handling a task, either modifying or retrieving data. The servers must continually communicate which tasks they are executing to ensure that all tasks are accomplished.
- Any server in the group can be modifying data, so a map of where the current version of any given record must be maintained. Before a given record may be read, its current version needs to be located.
- All the data is stored in a single, shared system. With large databases, this makes the individual files and indexes inherently larger. The larger files, combined with multiple server access, can make disk I/O a more difficult bottleneck to tune.

Shared all architectures will have a different set of areas to monitor and tune. The communications between servers is very critical to maintaining high performance. The communications to coordinate the individual servers tends to be chatty—lots of small packets of information moving between systems. A larger amount of memory must be dedicated to tracking the tasks that each server is performing. Handling the disk requests of multiple servers simultaneously must be carefully balanced.

The disk storage system must also be shared by all the servers. They all independently store and retrieve information from the same data files and transaction logs. This means the system does not inherently scale out access to the data. The disk subsystem can easily remain a bottleneck for the warehouse.

Just adding servers to a shared all architecture may not address its performance issues. When the problem is the communication between the individual servers or contention over access to the disk subsystem, additional servers may not provide much improvement. Sometimes, they can aggravate the issue. Understanding where the bottleneck is located is always crucial to determining how to remedy the problem.

Shared Nothing Architecture

Shared nothing architectures partition the data between individual data nodes. That allows each server in the system to concentrate on a smaller subset of the data. The tasks for each data node are clearly defined, so less communication between the servers is required. Disk I/O can be optimized for access from a single server, so many of the bottlenecks found in a shared all architecture are alleviated. Disk I/O can also be performed in parallel, with each data node storing or retrieving data within their individual subset of the data warehouse at the same time. This reduces contention and parallelizing operations speeds total throughput.

The art to optimizing a shared nothing architecture lays in the partitioning of the data. Carefully analyzing the most common queries can help distribute the data across the nodes to minimize the work of joining the data retrieved by each individual node once the final query is achieved. The proper distribution of data can also ensure that the workload distributes evenly across the nodes and that no one node becomes a bottleneck for the others.

The optimal results come when the individual data nodes return results to the central query node that requires minimal work to collate into the final result. The work required on the central node can often be minimized by replication of common lookup tables across all the data nodes and partitioning the data so that the results from each node can be appended to one another with minimal effort. Monitoring the workloads submitted to the system and tracking the balance of work across the individual data nodes can help determine whether the data has been optimally partitioned. Changes in the composition of the data or the most prevalent queries can lead to re-partitioning of the data.

Note

I worked on a system that partitioned data according to the region where individual stores were located. There were three regions and the balance between them was a good starting point. After the system had been running for a number of months, a small group of stores located in one region was purchased. This added a lot of data to one of the regions and placed more load on the data node that handled that region. When we saw that the workload across the data nodes was imbalanced and determined the underlying cause, we re-partitioned the data using districts (a more granular measure) instead of regions. That allowed us to restore the order within the system and kept performance optimized.

Data Compression and Performance

Data compression is always an interesting topic with large data warehouses. Even though the cost of disk storage continues to fall, the large amounts of enterprise class data storage required by large data warehouses remains a considerable expense. Compressing the data and using less storage can be a strong temptation.

Data compression stores the data in a smaller space by trading space for CPU cycles. For instance, if there are a number of null fields stored in a page, a marker can be placed at the beginning of the row that indicates which fields are null. Then a null character need not be stored for each null field in the row. When tables are wide and columns sparsely populated, this can save an amazing amount of space. There are a variety of techniques that can be used to compress the data that is actually written to and retrieved from the disks.

Resource

A white paper that explains techniques for database compression and their advantages can be found at http://web.cecs.pdx.edu/~len/compression.pdf.

Data compression can alleviate I/O bottlenecks. Reading and writing smaller amounts of data to the disks can relieve pressure on the disk I/O system. Allowing the compressed data to flow between servers can also reduce pressure on the network interconnections.

The cost of compression is paid by the CPU, and to a lesser extent, the system memory. To work with compressed data, the CPU must use compression algorithms to store the data in a more compact fashion. When the data is retrieved, it must be decompressed. Different types of data enjoy different levels of compression. Floating point decimals are very difficult to compress, while text strings containing language often enjoy compression ratios of 2 or 3 to 1. The type of data stored, therefore, will help determine the value of compression.

Operational data stores often contain more textual data as well as null field values. They will typically benefit from data compression. Analytical data structures, such as OLAP cubes and data mining model structures, primarily store numeric data. If the data is sparsely populated, they can benefit from compression, but when most of the columns contain data, they will not enjoy the same economies found in an operational data store.

Once again, the issue is balance. Most data warehouse systems have more excess processor capacity than network and disk I/O capacity. For these systems, data compression typically improves performance. Data compression will also reduce the requirements for disk space, which will help reduce hardware costs. If, however, the CPUs are near capacity and the network bandwidth and disk I/O have unutilized capacity, data compression may denigrate performance.

The value of data compression is also largely dependent on its implementation within the data warehouse system itself. If the data can be processed in a compressed state and not de-compressed until the final query results are delivered to the requestor, the entire system can work with a much smaller data stream. This can reduce pressure on most of the subsystems, including the CPU. If data can be joined and filtered by the query (performing exact matches, grouping, joins, and so on) before decompression, smaller streams of bytes can be passed through the disk I/O, network, and memory subsystems. The data need not be decompressed until the final result is materialized. Understanding how the data warehouse system handles decompression will help you understand its benefit to that system.

Query Processing Challenges

Getting a reasonable response time when querying a table with billions of rows is always a challenge. Using the system's internal mechanisms to help speed the query process is obvious but frequently overlooked. With large scale data warehouses that distribute data across multiple servers and possibly multiple independent disk subsystems, there are new tools and techniques available. By understanding how the system in question handles a query, the elements of the query can be broken down and the individual tools at the disposal of the operations team can be used to provide the best possible performance.

Building and Managing Data Retrieval Aids

For the purposes of this discussion, data retrieval aids are any data structures built and maintained within the data warehouse that help return results to queries faster. They are often de-normalized structures that copy data and relieve the need to perform joins, pre-aggregate data, or build models from analyzing data in detail.

Denormalized Relational Data Structures

In relational data structures, the most expensive retrieval operations involve scanning the contents of a table, one row at a time, to find the requested data. When tables are billions of rows, this becomes a very serious issue indeed. The solution is often found in indexes and materialized views.

Indexes can help locate data by providing an ordered subset of the data. This much smaller set of data can help locate desired rows faster because the data structure itself is smaller. If the index is in the same order as the query, it can also minimize the need for sort operations. Often, only a small amount of data is required from a table that is used in a join, such as looking up a product name based on the product ID. If the product name is part of the index, the products table need not be retrieved (this is called a covering index).

Indexes are helpful when they are small and can help reduce the amount of time spent going to the parent table. They do take space to create and system resources to maintain. They should be planned carefully to provide the greatest savings for the effort expended in their creation and maintenance.

When building indexes for shared nothing architectures, it is important that the index be available to all the data nodes so that they can take advantage of the performance boost. By replicating the index to all the associated data nodes, the index is local to each node. This can help resolve queries within the boundaries of the data node itself without communicating to the other nodes. This often reduces internode communications and can improve performance.

> **Note**
>
> I worked on a system where the designer had indexed every column in one of the largest tables in the data warehouse, hoping to improve performance (he had read that indexes were good in data warehouses). Most of the indexes were never used and made no sense to maintain. We analyzed the table usage and dropped all but three of the indexes. The queries performed the same, but processing time and space consumed in the data warehouse dropped. Although most cases are not this extreme, it makes the point that any structure built in a data warehouse should be built for a purpose and maintained only as long as it fulfills that purpose.

Joining tables on the fly, in response to a query, is common. It can also be expensive. If a query is performed on a regular basis and requires multiple joins, a materialized view can improve performance. The materialized view will perform the join once and write the results to the data nodes. Querying the view becomes a simple matter of retrieving rows from a single table.

The cost of the materialized view is the resources required to maintain it. Every data load will cause the data within the view to be altered to match the revised data. Time and resources to process the view must be accounted when counting the cost. Shared nothing architecture helps mitigate this cost, to a degree, by parallelizing the effort to maintain the view across the data nodes.

The other cost of materialized views is that the view is linked to the schema of the underlying component tables. If a column is altered or deleted in one of the base tables of the view, the view itself must be altered to match the change. This is not a high cost, but ignoring it can break the applications.

Aggregate tables are also commonly used to store pre-calculated and pre-joined query results. Very similar to materialized views, the aggregate tables require processing to update the data and disk I/O to write the results. They are beneficial if the resulting queries can justify the cost to the system in resources expended.

OLAP Cubes

One of the most common aids for querying quantitative data that contains trends is the use of OLAP data structures. The acceleration enjoyed by OLAP cubes is realized by aggregating the numeric data prior to retrieving it ("processing" the cube). The key to getting the most benefit from the cubes, then, is the design of efficient aggregations.

The aggregations are based on the design of the dimensions. For instance, store sales will have a store dimension. The transactions will take place at a store. The store is part of a district; the district is part of a region; the region is part of a division. The sales amount of the individual transaction can be aggregated with all the other transactions for the store for that day: the store sales can be totaled as part of the district sales, the district sales as part of the region, and so on.

As more dimensions are added, you can design aggregations that cross-correlate with the other dimensions. Thus, there might be store sales by product, by product sub-category, by product category, and so on. The more dimensions that are contained in the OLAP cube (customer, marketing program, store location) the more potential aggregations exist. Cubes run most efficiently when the OLAP queries can select from a single aggregation table to retrieve results. Finely tuned cubes will have aggregations that minimize the number of rows required to retrieve a specific request.

Note

Real-time or near real-time OLAP analysis requires continuous updating of the OLAP aggregation tables as the data is entered continually into the data warehouse. I have worked on commodity trading systems that used near real-time OLAP cubes to guide their traders based on activity throughout the day. A large scale data warehouse system that has enough capacity to load data and update the aggregation tables while remaining responsive to the users provided this firm with a competitive trading edge. To fine tune that edge, we carefully mapped the aggregations to the specific range of queries used by the traders. Longer-term analysis of trends was done in different OLAP cubes that we updated nightly rather than continually.

Conversely, the aggregation tables can take some time to process. If a cube has 150 aggregation tables, adding or updating a single fact can touch many or even all of them. Determining which aggregations will provide the greatest improvements in frequently used queries and eliminating aggregations seldom accessed will help optimize the time spent processing cubes and provides the best benefit from the system.

There are advantages in large scale, multi-server systems in handling the aggregation process. Parallelization of the aggregations will help reduce the time spent building the aggregations. Distributing the resulting aggregations across the nodes will also help parallelize data retrieval. Also, a shared nothing architecture allows each server to deal will a smaller amount of data. This helps the server run more efficiently.

In-database analytics also play a role in improving the system. A fairly common operation in retrieving data from OLAP cubes is moving through the aggregation to find a set of data that is interesting (either because things are going well or things are going not so well). The user will often want to drill through to see the details on the records that made up this aggregation. Having the data for the detail in the same database as the aggregations makes it much simpler and more efficient to retrieve these records. The systems that can support all the database data in a single unit and move seamlessly from aggregate OLAP data to the granular relational data stored in an ODS will better support this type of querying.

Data Mining Models

A data mining model uses statistical or heuristic analysis to find patterns in data. The patterns can be used for predictive analysis of data. The key to data mining is to find a pattern that is distinctive and consistent within the data. Once the pattern has been found and verified, it can be used to predict probable outcomes based on the data at hand.

The key to creating accurate data models is providing sufficient data to the model. A set of data is used to create the parameters that shape the initial pattern. Then a second subset of the data can be used to evaluate the accuracy of the model. The more representative data that is available for analysis, the more accurately the model can be refined.

Large data warehouses contain sufficient data to create these models. However, the model needs to work through quite a bit of data to be refined. The cost to develop the models lies in retrieving the actual data and running it through the algorithm to adjust the internal parameters and refine the accuracy.

The models themselves, then, use the refined data point to work their magic and predict results. Based on known data (independent variables), they use the model to provide the most likely values for the unknown data (dependent variables). This requires access to the model and a fair amount of processing power where the queries are executed. Providing efficient access to the data for training the resulting model data structures will make this process much more responsive.

Data Marts

Data marts help focus information. Once the data from key data sources has been collated from throughout the organization, smaller subsets of the data that concentrate on specific subject areas can be built. Data marts are commonly used with enterprise data warehouses to help provide the data to answer the specific questions of a specific group. For instance, customer service may need data about the order patterns, payment patterns, marketing exposure, and retail outlets frequented by their clientele. Rather than search through the entire data warehouse for such information, the pertinent data can be extracted from the larger data warehouse. This provides a smaller, more carefully focused body of information to query. The data in a data mart may be relational, analytic, or both.

Operationally, data marts need to be synchronized with the enterprise data warehouse through ETL processes. They may be hosted within the primary data warehouse system or the data may be transferred to a separate reporting server or server cluster. Frequently, OLAP cubes are developed from this data. The processes used to maintain data marts are very different than most direct user queries. They are often involved with automated processes and can put a significant load on the primary data source tables while the data is being extracted to update the data mart.

The parallelization of querying the primary data stores to produce the data within the data mart provided by a large scale data warehouse will make this process faster. The capacity of the data warehouse and its ability to move data effectively within the internal network of servers can help build the internal structures efficiently while serving the other requirements placed on the system.

Optimizing Queries to Utilize the Platform

The design and implementation of the database should work hand in hand with the queries that are used to store and retrieve data. Operationally, working with large data tables and a multi-server platform add some wrinkles to the process.

Most database professionals know the typical things that should be avoided when creating queries against a database. Queries that return most of the rows in the database are slow. Joins against tables without the use of indexes typically respond poorly. Cross-joins can create runaway queries that can consume inordinate amounts of server resources. With large scale databases, these issues are magnified by the sheer number of rows contained within the database itself.

Multi-server databases have unique characteristics that can be used to improve performance. For example, in a shared nothing architecture, each database node handles a subset of the data contained in the entire database. The system may have one or more dedicated ETL nodes, so those nodes can be used to prepare the data that will be loaded into the node in advance; the system may also perform loading through nodes used for query or results processing. In the former case, preparation of the update will not affect the nodes that are responding to user queries. The data can be prepared to load into each data node as efficiently as possible. This minimizes the disruption created by data loading.

The manner in which data is partitioned across the nodes also affects query response. Each query is broken down into the sub-tasks of the query plan. The more completely the tasks can be executed on a single data node, the less work will need to be accomplished to collate the results from each node within the system. For instance, if two tables are joined, and all the rows for the two tables that the data node requires are contained on that individual node, it can perform the join and send the results back to the central processing node. If some of the rows are located on one node and others are contained on another node, the partial results of the query are returned to the node so that the join can be performed. The additional steps consume network, CPU, and memory resources.

You cannot perfectly align the data for every table to every data node. The product table may be aligned by product category, the sales table aligned by store. Sometimes the data will align, sometimes it will not.

You can help the issue somewhat by replicating some tables. Common lookup tables used in most frequent queries can be replicated in their entirety across all the nodes in the system. Although this setup will consume space, it will allow many queries to resolve within the bounds of each individual node.

You can also make certain that all the nodes are evenly engaged. Some partitioning schemes could heavily favor one node, leaving the other nodes idle. Consider partitioning schemes that balance the workload across all the data nodes. Although you cannot accommodate every query, you can balance the highest demand queries and find schemes that help the system's overall performance.

Using a multi-server system will provide new areas of performance management to consider, but the impact of performing parallel operations on multiple servers provides a dramatic increase in the overall performance of the system. It allows the complete system to scale to a much larger level than a single server can realistically achieve. This allows the data warehouse to grow and enables benefits like in-database analytics to become realistic.

Monitoring and Tuning the Workloads

Optimizing a single query is relatively simple. The art of database tuning involves optimizing the entire workload. Adding indexes often improves queries at the expense of disk space and processing resources during loads. When does the advantage of the improved query justify the cost of maintaining the index?

There is no simple answer to this question. The best approach is to consistently monitor the load placed on the server and determine which queries present the greatest load to the server over a period of time (be it a day, a week, or a month).

With large scale enterprise data warehouses that perform ETL, data storage and retrieval, and processing of analytical data, the workload is very broad. The system should be set to capture a representative load of data that it processes during the tuning period. The total time spent on each query or load operation can be logged and rated. The process, then, is to work through the list from the queries that present the largest cumulative loads to those that present the smallest.

> **Note**
>
> I have found that it is the greatest total cumulative workload that really matters. I worked on a system that had a query that took more than 7 hours to process. It performed table scans on three large tables and had to join all of their rows in order to satisfy the query. The client had us invest more than 200 hours optimizing the query, and we got it down to 5 ½ hours. Upon further analysis, we learned that the query was only run once a month. There was another query that ran in 50ms. 30 minutes of investigation discovered a simple index that reduced that query to 22ms. The query was run on average 150 times an hour. Where was the time better spent? What provided the best overall performance in the operation of the system? It is total load, in context with the entire workload, that should guide you into determining where to spend time optimizing.

Changes should be made gradually and measurements taken to ensure that there is a net overall improvement in the system. It is easy to add or remove a materialized view and miss some way in which that view is used—or miscalculate the effort to maintain it. If too many changes are made in a measurement period, it may be too difficult to isolate which changes were effective and which were not.

Tuning, therefore, should be an ongoing part of the operational plan. The system should be continually logging its activities and the operation staff should be reviewing those logs regularly to find ways to improve the system.

Monitoring the System

The operations staff needs to know that the data warehouse is operating and meeting the needs of the client base. The team should look for areas where the system is beginning to slow down and endanger SLAs for operation. It means building a basis for proactively looking for changes that will improve system performance. It means ensuring that the system has been secured in the unlikely event of a system failure.

Identifying the Bottlenecks

We have previously discussed the various activities that the entire warehouse system needs to perform. It must ETL enterprise data from multiple sources. It must store and retrieve that data in a timely manner. It must internally process that data to build reporting and analytical structures.

All of these activities compete with one another within the system. When the system as a whole slows, the root cause of that slowdown needs to be identified and quickly dealt with. System monitoring should allow the processes within the system to track the performance of activities over time. By monitoring and logging the activity, shifts can be identified and linked back to root causes. This will help tune the system.

One of the advantages of a comprehensive data warehouse and analytic system is that the performance measurements are all captured against that particular system. It greatly simplifies the process of seeing how changes in the ETL system affect demand in the data storage engine. The same resources that retrieve report data also process the OLAP cubes, so it becomes clearer when these operations should be scheduled to avoid conflicting with one another.

> **Note**
>
> I worked on a project that captured the data from an ETL system, a database that housed a warehouse, an OLAP system, and a reporting system. Each system was hosted on a different server, and we needed to correlate their combined activities to determine which system was struggling. We could no longer process data within our nightly processing window. The collection of the data became a data mart project within itself, just to discover which system was not meeting our expected levels of performance.

Monitoring also needs to immediately capture failures and notify the operations staff so that system faults can be corrected. If an ETL job fails, or a network card goes offline, the operators need to be notified so that they can correct the issue and address it before the issue harms the business.

Proactive Monitoring of Change in Performance

Over time, the performance of the system will change. This can come from many sources:

- Data volume provided by a source system can grow, adding time to load data and distribute it into the system.
- Data sources change as systems are replaced or new systems brought online. This changes the demands on the ETL system and may change the characteristics of the workload placed on the server.
- Mergers and acquisitions can make rapid, large scale changes to the data stored in the data warehouse and processed through the analytic systems.
- Regulatory changes can alter the level of detail or longevity of data stored within the system.

These changes can cause increased demand on the system gradually over time or suddenly when dramatic changes occur within the organization. The operations staff needs the ability to resize the system to meet the changing needs of the enterprise.

> **Note**
>
> One of the advantages of operating a data warehouse with analytic processing capability is that the system can be used to monitor its own performance. I have loaded performance log information into the system and used the data to build OLAP cubes. The data can show trends within the workloads. I have used this information to project the rate at which demand for system resources is growing and provide management with the hard data they need to justify expansion of the system resources.

Some systems make this process simpler than others. For instance, a shared nothing architecture can often scale out to add processing power and disk capacity. Adding data nodes often has a nearly linear increase in capacity, which makes the system easy to resize when the demand grows. That means the organization can purchase the size system they need at the beginning and grow the system as its usage and demand grows.

Reacting to Loss of Continuity

As businesses avail themselves more of the data contained within the data warehouse, it becomes increasingly hard to do without. The system should be designed to protect the organization from loss, whether it be loss of data or loss of productivity if the data is unavailable to the business decision makers.

Multi-server data warehouse systems provide good internal resiliency against individual server failure. The individual nodes within the system can be protected with failover nodes. Should one node fail, the backup node will seamlessly take over the workload with little interruption in service.

The system often contains data that would be difficult, if not impossible to re-construct from the source data from which it was extracted. That means the system should be backed up on a regular basis.

In a shared nothing architecture, a dedicated backup node can be used to collect the data and create the backup. This minimizes the effect on the other nodes within the system, accessing them only to rapidly collect their data for backup. The data is staged by the backup node and prepared to be archived offline.

Restoration of data can also be accelerated. The data nodes can load data in parallel, so reloading backup data into the nodes can be done quite rapidly without the contention that would be encountered in a system that stored its data in much larger single shared files. The combination of multiple HBAs, disk arrays, and processors to handle the restoration makes the data available to the users sooner.

Summary

This chapter discussed many of the operational considerations involved in working with large scale data warehouses and analytic systems. We explored getting data into the system, storing and retrieving the data, optimizing the queries, and keeping the system operating effectively. In the next chapter, we will explore analytic considerations for large scale data warehouses.